CSCI 3753: Operating Systems

Week 5
September 26, 2025

University of Colorado **Boulder**

Announcements

- PA3 due Sunday at midnight!
- Quiz 5 due today at midnight
- Interview grading slots are now available on Canvas
  - Sign up BEFORE Tuesday
  - As the interviews get closer, the slot availability will lessen!
- As always recitation materials are stored here

Recap

- PA0,1,2 ✅
- Problem Set 1 ✅
- PA3 🔁
- Introduced:
  - C stdin + stdout

PA 3

- create a Device Driver Module (LKM)
- implement file operations
  - open, seek, read, write, release
- make and load the module
- create a Device File for this Device

PA 3 🔁

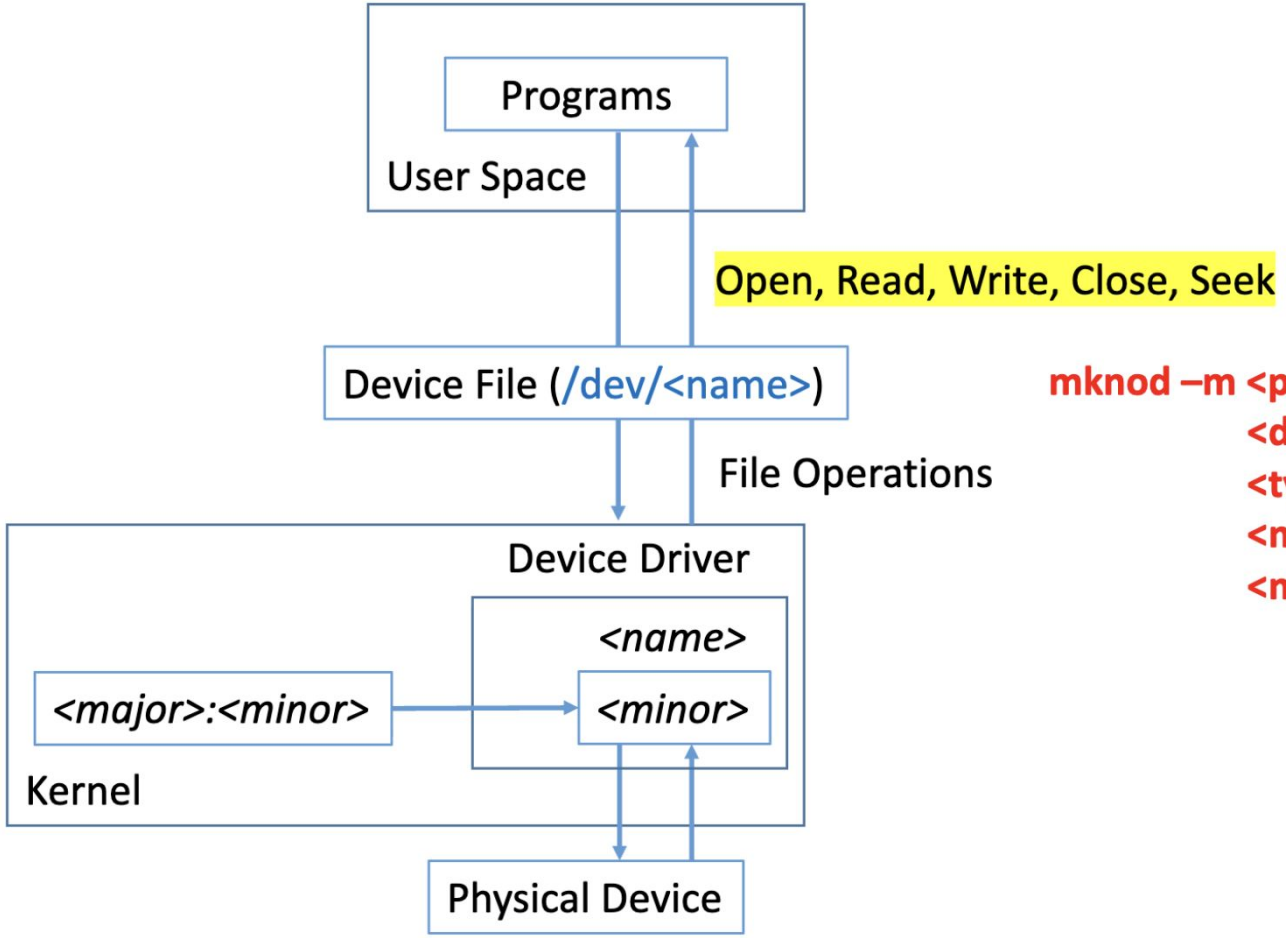
- create a Device Driver Module (LKM)
- implement file operations
  - open, seek, read, write, release
- make and load the module
- create a Device File for this Device

- **create a test program**

PA 2 ✅

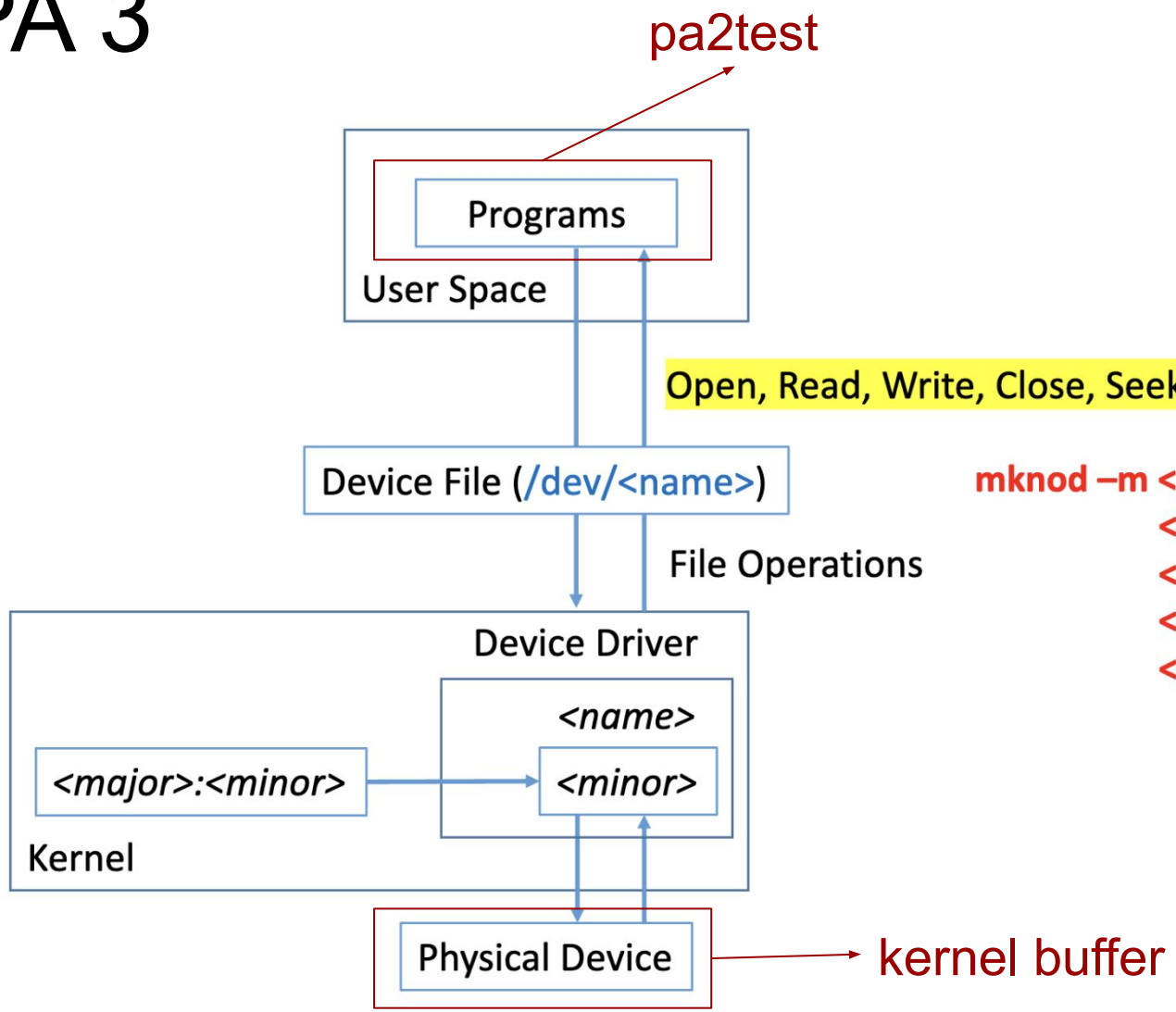
test with: ./pa2test /dev/pa3_char_device

PA 3



mknod –m <permission>
<device_file_location>
<type of driver>
<major number>
<minor number>

PA 3



mknod –m <permission>
<device_file_location>
<type of driver>
<major number>
<minor number>

PA3 – Character Device Driver (LKM) Flowchart

High-level data/control flow for a simple character driver that stores bytes in a fixed in-kernel buffer and exposes `open/read/write/llseek/release` via `file_operations`. Use this with the PA2 tester and a device node in `/dev`.

Syscall/VFS path | Driver logic | Setup/Teardown

1) Build & Load LKM
`insmod pa3_char_driver.ko` → *module_init*

2) Register Char Device
`register_chrdev(511, "pa3_char_device", &fops)`

3) Allocate Buffer
`kmalloc(900, GFP_KERNEL)` → *device_buffer*

4) Create Device Node
`mknod /dev/pa3_char_device c 511 0`

PA2 Tester / Shell
`open/read/write/lseek/close` on `/dev/pa3_char_device`

VFS & Dispatch
Kernel routes ops via `struct file_operations`

Driver: file_operations
`.open .read .write .llseek .release` → your handlers

Device Buffer
Fixed 900 bytes in kernel memory

open()
Inc counter; `printk` usage.

read()
Clamp to end; `copy_to_user`; advance `*ppos`; return bytes read.

write()
Clamp to space; `copy_from_user`; advance `*ppos`; return bytes written.

llseek()
SET/CUR/END → compute new pos; clamp [0,900]; return new pos.

Edge Cases
Partial R/W at buffer end; SEEK before start or past end clamps; zero-length ops okay.

Observability
`printk` bytes & counters; verify via `dmesg`, `/proc/devices`.

Unload LKM
`rmmod pa3_char_driver` → *module_exit*

Unregister Device
`unregister_chrdev(511, ...)`

Free Buffer
`kfree(device_buffer)`

Remove Node
`rm /dev/pa3_char_device`

Quick tips: Match `#define MAJOR_NUMBER 511`, set `BUFFER_SIZE 900`, and point your `mknod` to that major/minor. Use your PA2 tester on the device node to exercise partial reads/writes and clamped seeks.

PA3 Demo

- Available on Canvas
- Be sure to install it on your generic kernel

PA3 - Functionality

- module_init
- modude_exit
- open
- release
- read
- write
- llseek

PA3 - Functionality

- module_init
  - Announce the module is loading
  - Allocate the kernel buffer
  - Register the char device with the kernel
  - If anything fails, clean up memory and print error
- modude_exit
- open
- release
- read
- write
- llseek

PA3 - Functionality

- module_init
- modude_exit
  - Announce the module is unloading
  - Deregister the char device
  - Free the kernel memory
- open
- release
- read
- write
- llseek

PA3 - Functionality

- module_init
- modude_exit
- open
  - Increment open counter
  - log open count
- release
- read
- write
- llseek

PA3 - Functionality

- module_init
- modude_exit
- open
- release
  - Increment close counter
  - log close count
- read
- write
- llseek

PA3 - Functionality

- module_init
- modude_exit
- open
- release
- read
  - Look at the caller's current file position
  - If at end of buffer, indicate there is nothing left to read
  - Otherwise, determine how many bytes are actually available
  - Copy that many bytes into the caller's buffer in user space
  - Advance the caller's file position by the number of bytes delivered
  - Report how many bytes were delivered, or report error
- write
- llseek

PA3 - Functionality

- module_init
- modude_exit
- open
- release
- read
- write
  - Look at the caller's current position
  - If at end of buffer, indicate there is no room left
  - Otherwise, determine how much space remains
  - Copy that many bytes form the caller's user space buffer into the device buffer
  - Advance the caller's position by number of bytes stored
  - Report how many bytes were stored, or error
- llseek

PA3 - Functionality

- module_init
- modude_exit
- open
- release
- read
- write
- llseek
  - Depending on reference point:
    - Set the new position to an absolute value from start
    - or add an offset to the current position
    - or add an offset to the end of the buffer
  - Clamp the resulting position
  - Update the file position and report back
  - Seek should **never** fail

PA3 - Installation

- Available on Canvas

PA3 - Installation

1. Write LKM code
2. Add module to Makefile
   a. obj-m:=pa3_char_driver.o
3. Compile the module to get the .o file
   a. make -C /lib/modules/$(uname -r)/build M=$PWD
4. Insert the mod into the running kernel
   a. sudo insmod pa3_char_driver.ko
5. Create device node
   a. sudo mknod /dev/pa3_char_device c 511 0
   b. sudo chmod 666 /dev/pa3_char_device

Same as PA1

PA3 - Deinstallation

- Remove node:
  - sudo rmmod pa3_char_driver
  - sudo rm -f /dev/pa3_char_device

PA3 - Testing

- Use PA2
  - ./pa2test /dev/pa3_char_device

Looking Ahead

- PA4
  - Create a shared array
- PA5
  - Create a DNS resolver
- PA6
  - Combine these to create a multi-threaded DNS resolver

Looking Ahead

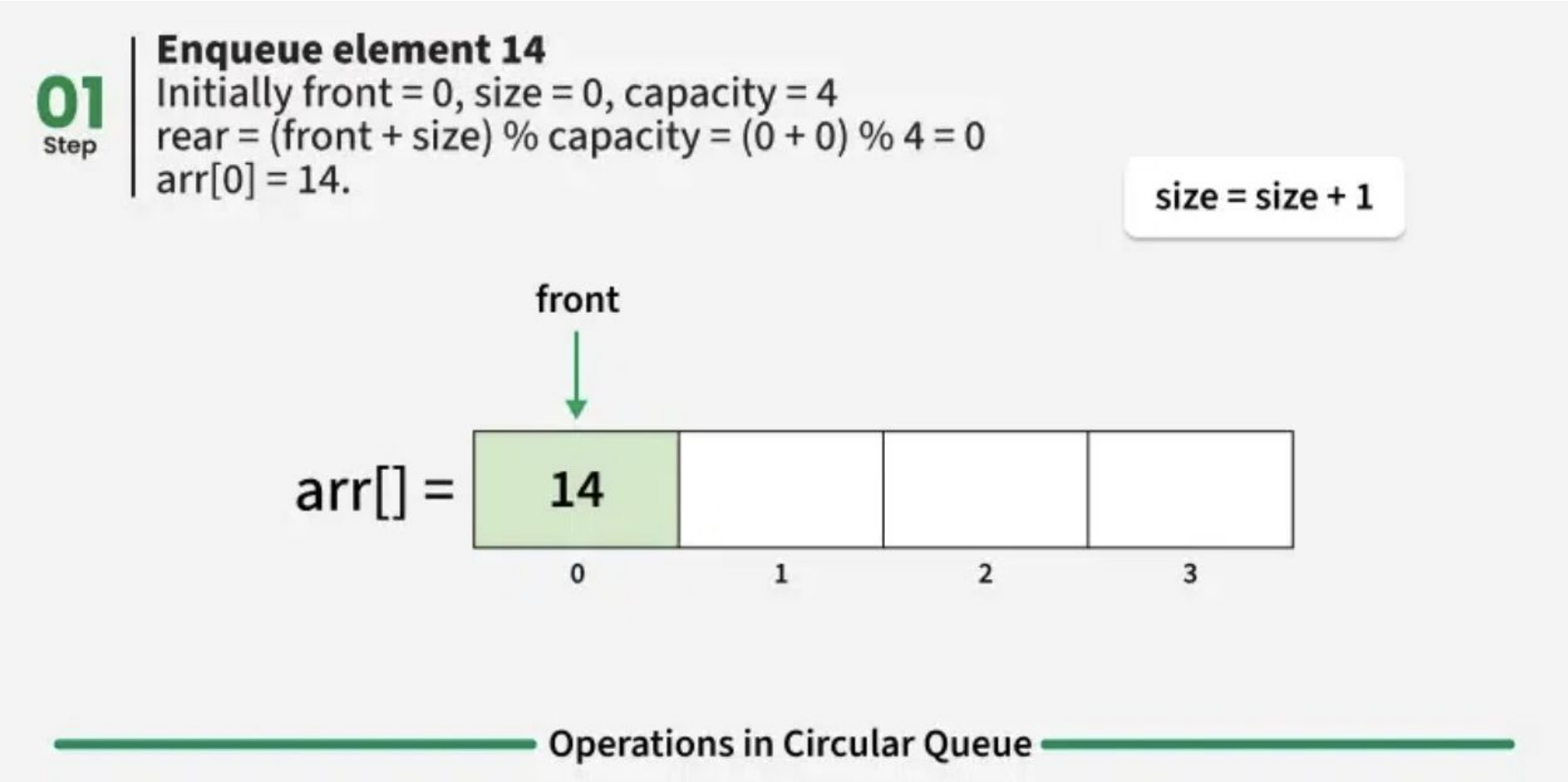
- PA4
  - Create a shared array
- PA5
  - Create a DNS resolver
- PA6
  - Combine these to create a multi-threaded DNS resolver

Bounded Buffer Problem!!

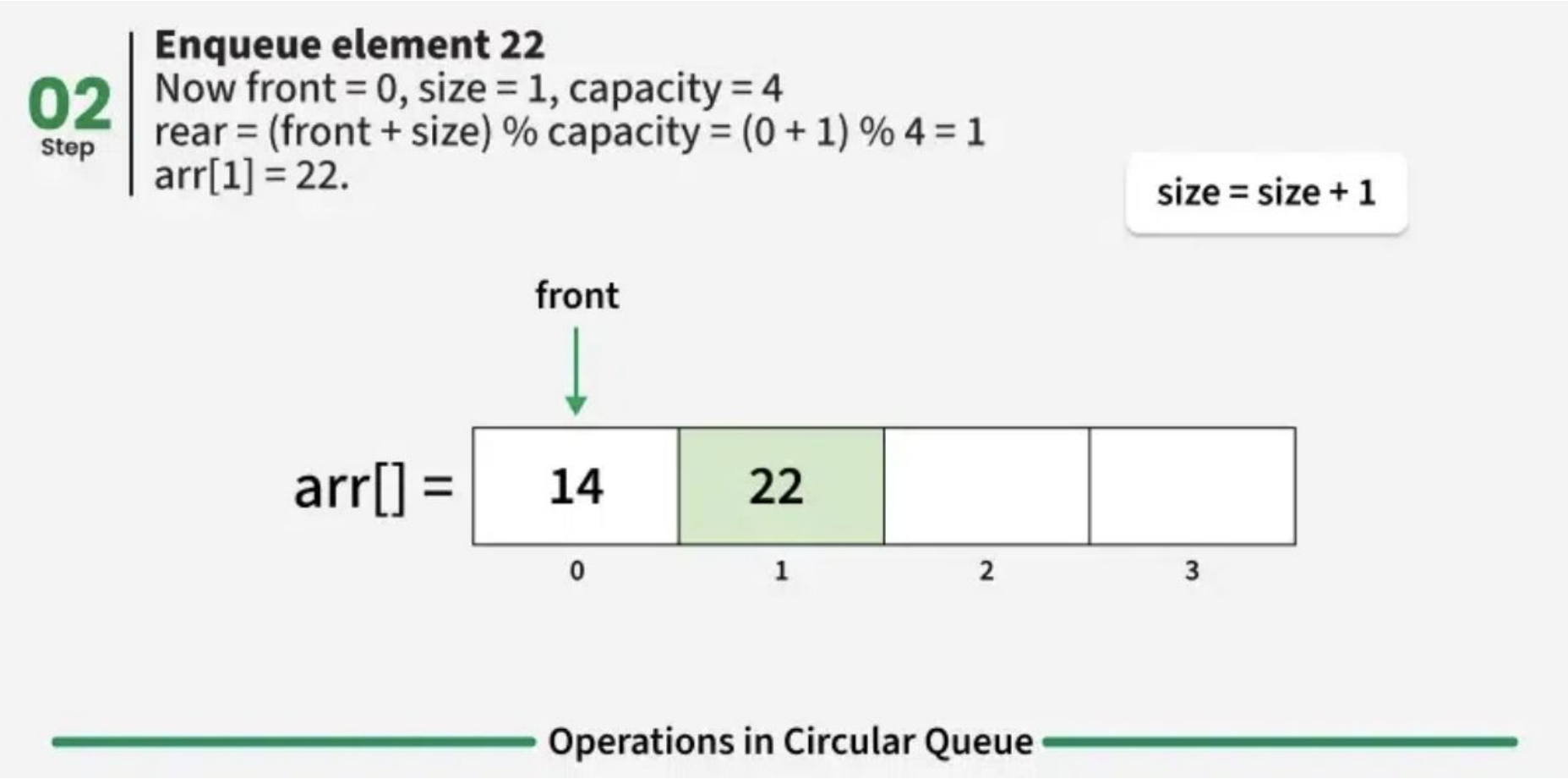
Looking Ahead

- PA4
  - Create a shared array
    - Must be:
      - thread safe!
      - built on top of one (or more) contiguous, linear memory arrays
    - Could be:
      - FIFO
      - LIFO
      - Circular queue
    - Cannot have:
      - linked lists, dictionaries, trees, other pre-built data structures in C
  - Turn in both .h (header) and .c files for your shared array

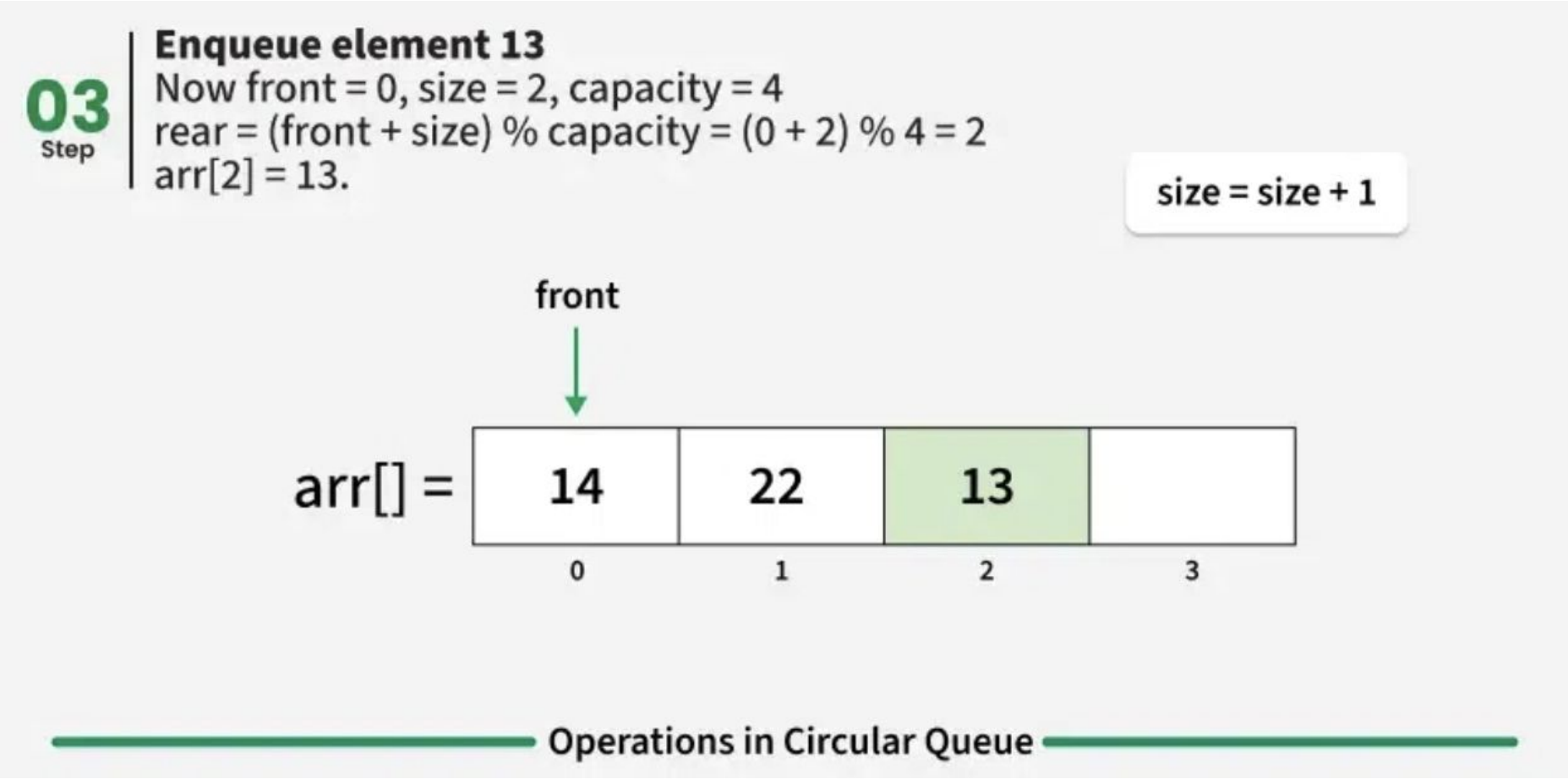
Circular Queue



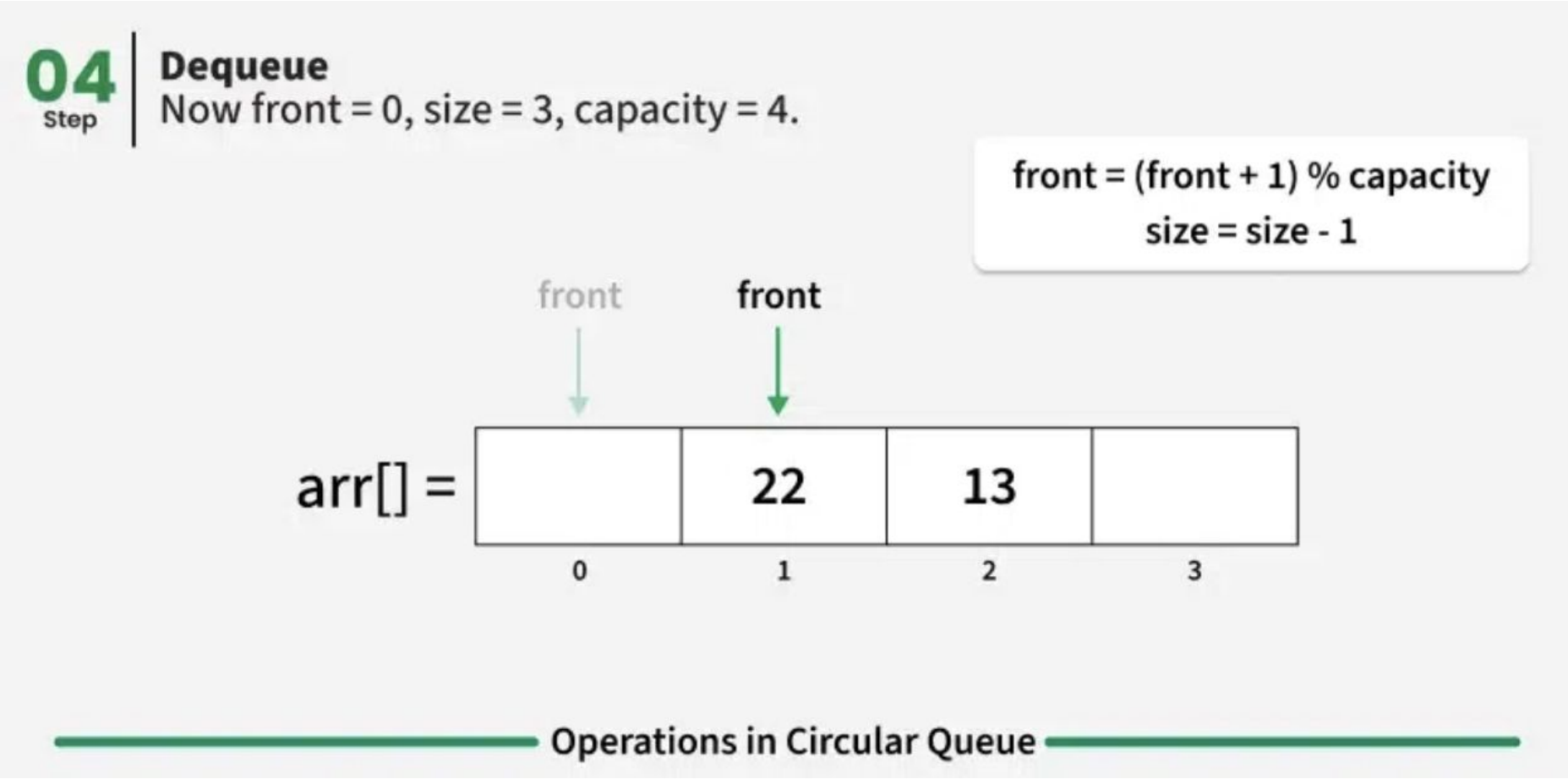
Circular Queue



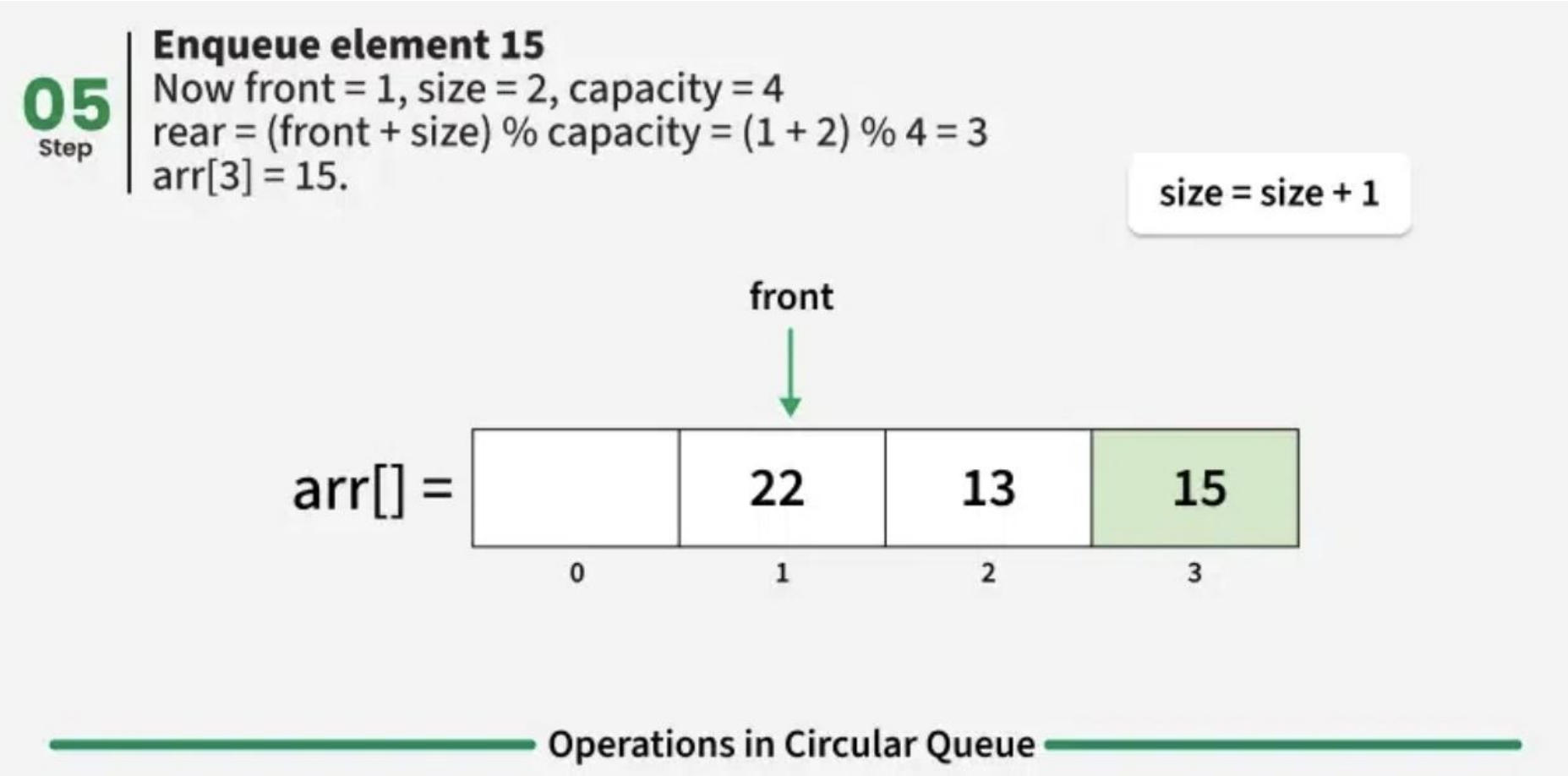
Circular Queue



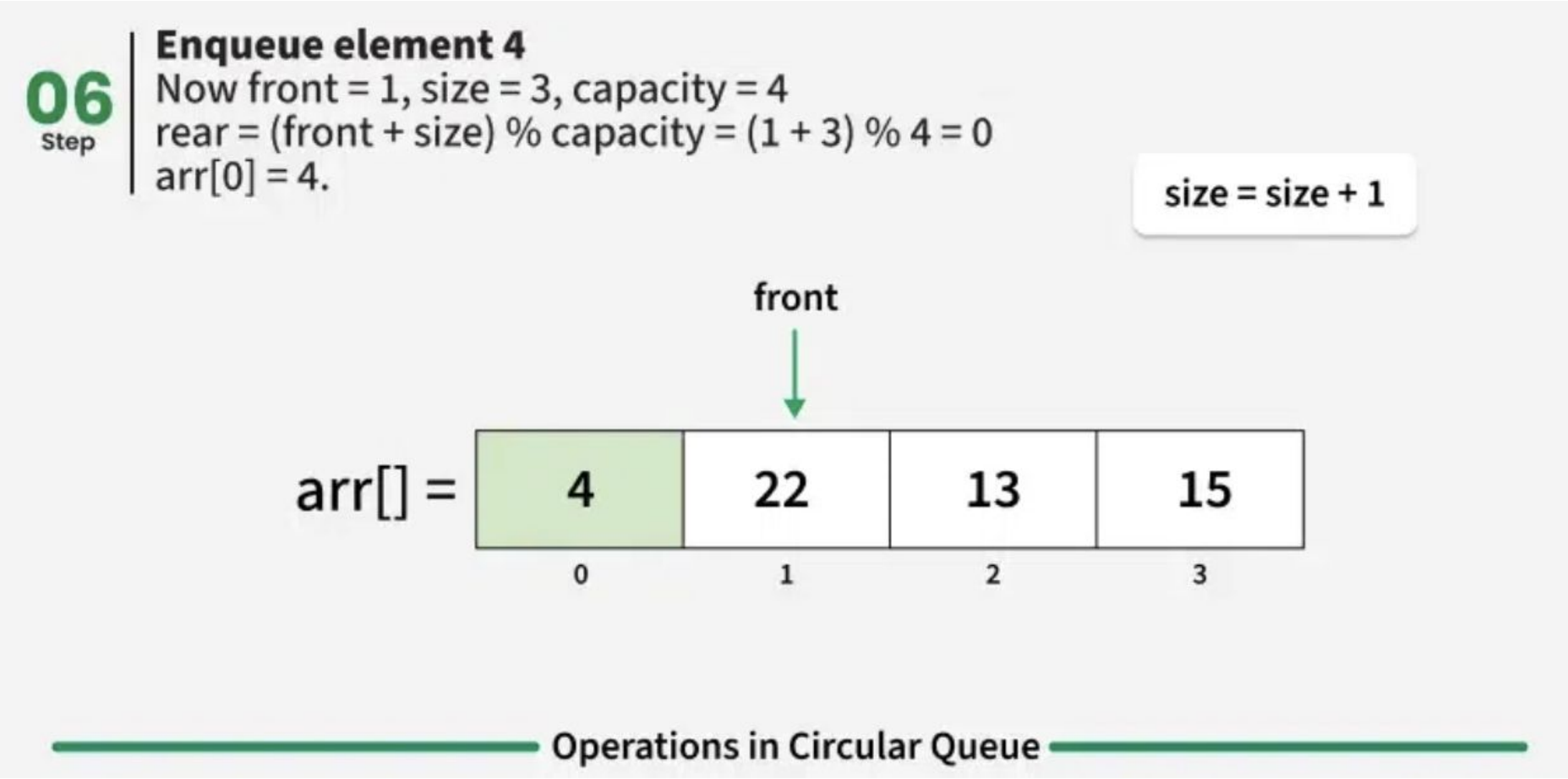
Circular Queue



Circular Queue



Circular Queue



Thread Safe Options

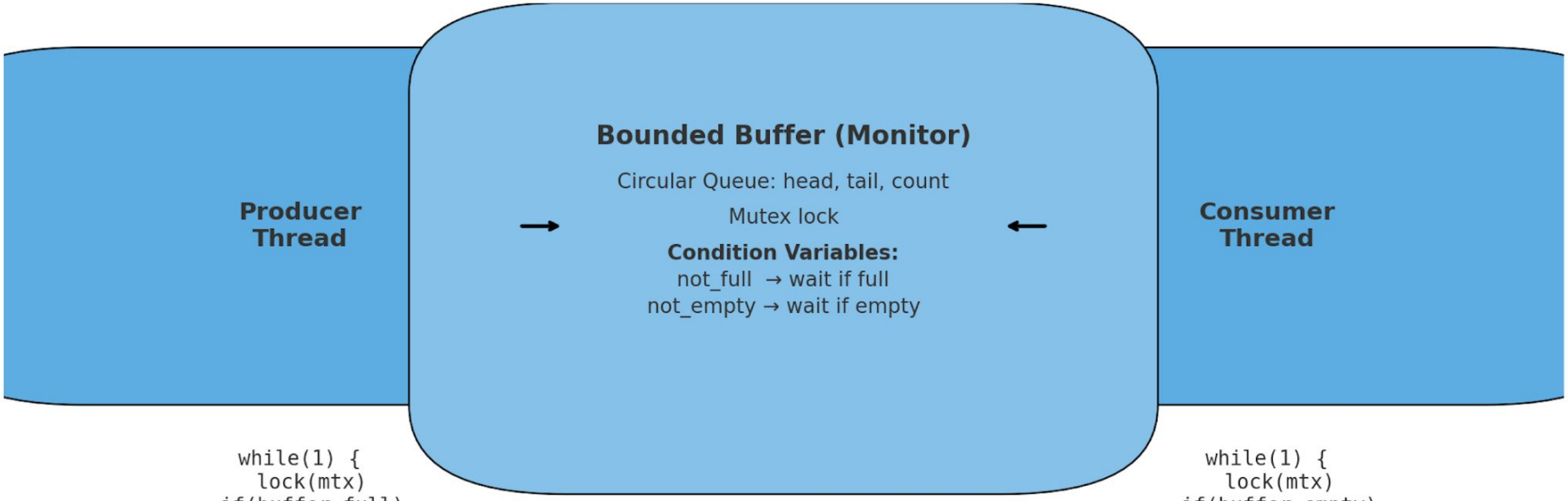
- Mutex Locks
- Condition variables
- Semaphores

Thread Safe Options

- Mutex Locks
- Condition variables
- Semaphores

Combine these two for the Monitor approach

Monitor Approach




```
while(1) {
  lock(mtx)
 if(buffer full)
wait(not_full, mtx);
 put item; count++
 signal(not_empty)
   unlock(mtx)
       }
```

```
while(1) {
  lock(mtx)
 if(buffer empty)
wait(not_empty, mtx);
 get item; count--
 signal(not_full)
   unlock(mtx)
       }
```

Note: wait(cond, mtx) releases the mutex and suspends the thread until signaled. This is NOT busy waiting — the thread sleeps and consumes no CPU while waiting.

C Thread Safety Demo

- Download it here!